**EE239AS Project 1**

**Regression Analysis**

**Student1**: Xiangrui Liu (004516858) **Student2**: Qi Wang (604522309)

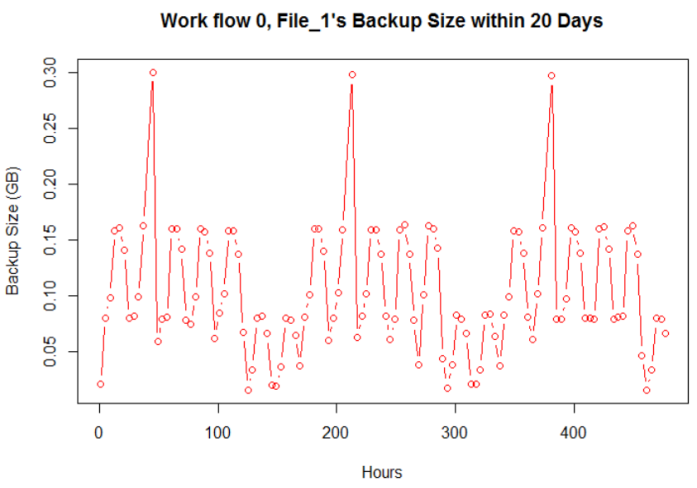
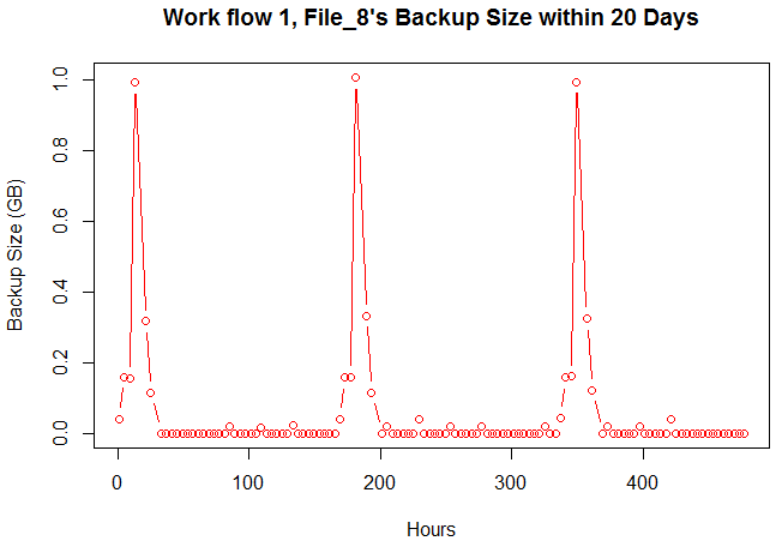
**Student3**: Qinyi Yan (704406413) **Student4**: Ziyin You (404412651)

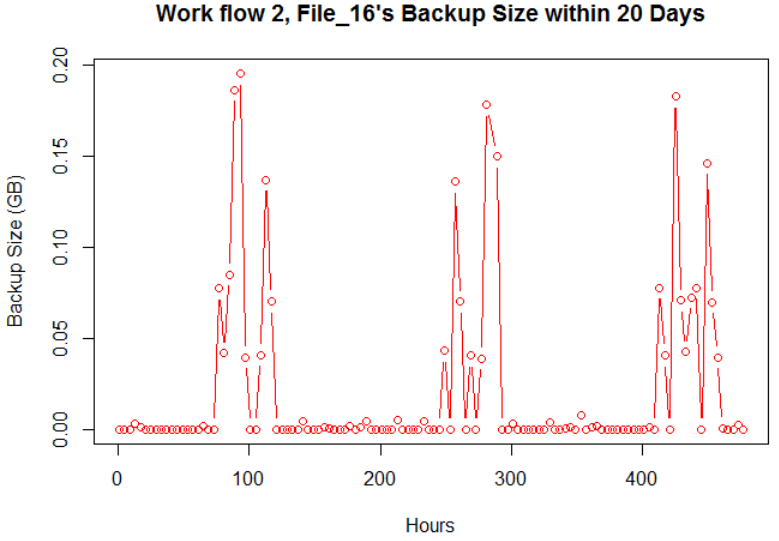
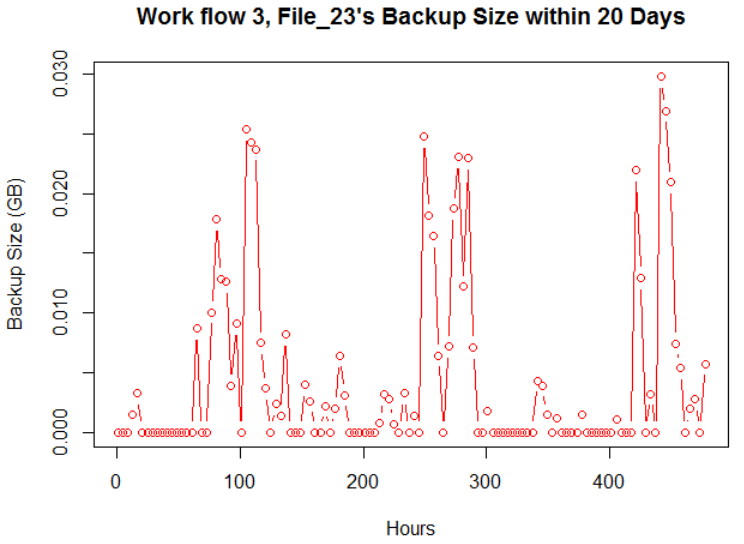
**Network backup Dataset**

*Question.1*

First of all, the Week in the data is converted from "Monday", "Tuesday"... to "1", "2" ... "7" in order to calculate the total backup time in *Hour*.

Then, for each of the Workflow, only one File Name which belong to that Workflow is chosen (since the trend is similar within the same Workflow). Then the Backup Size (GB) versus Backup Time (hour) is plotted for each of the Workflow as shown in Fig.1:

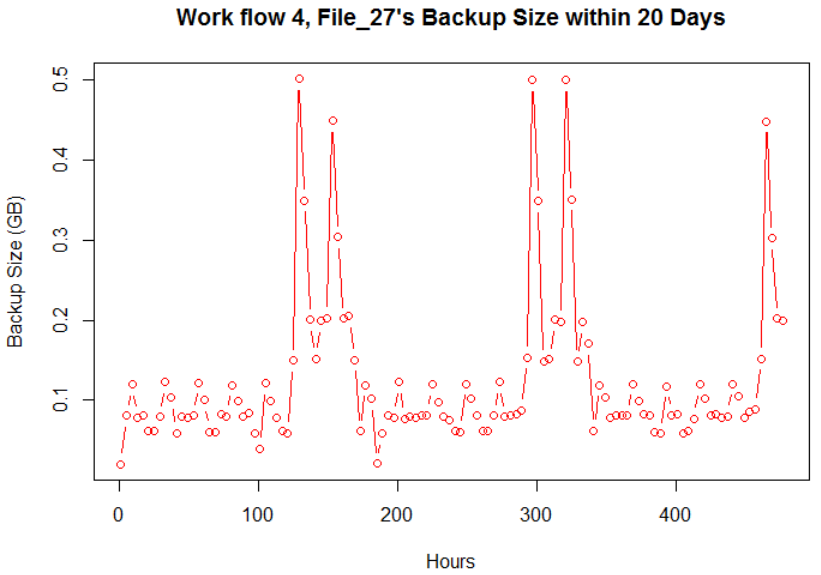


Figure.1 Backup Size (GB) versus Backup Time (hour)

For Workflow1, the data is repeated roughly every **150** hours.

For Workflow2, the data is repeated roughly every **140** hours.

For Workflow3, the data is repeated roughly every **160** hours.

For Workflow4, repeating pattern is not very obvious..

For Workflow5, the data is repeated roughly every **160** hours.

*Question.2*

*2(a) Linear Regression*

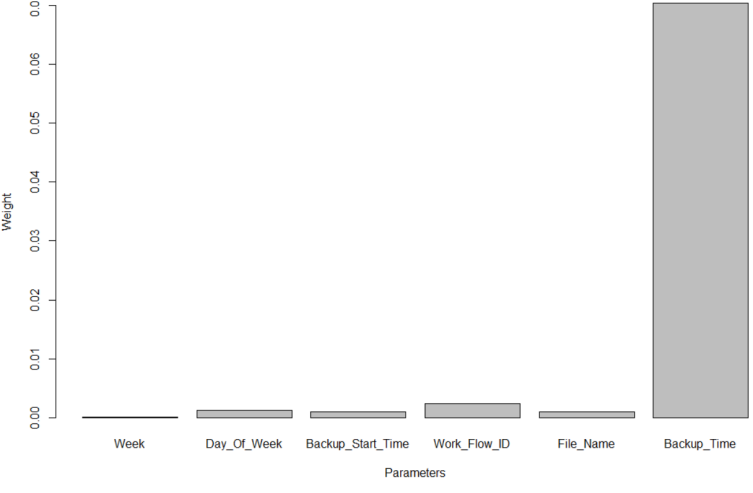
The linear Regression calculation flow is as follows:

First of all, the data is randomly divided into 10 folds. Then, one of the fold is chosen to be testing group, and the other 9 folds are chosen to be training groups. We can get the fitting coefficients by using the function below:

lm (Size\_of\_Backup ~ Week+Day\_Of\_Week+Backup\_Start\_Time+Work\_Flow\_ID+File\_Name+Backup\_Time, data=training)

Then using the fitting coefficients, we can predict the Backup Size of the testing group, compare the predicted data with the actual data, and calculate the RMSE for the current fold. Finally, the average **RMSE** can be calculated, which is **0.0789572315048197**.

Also, the coefficients of the linear regression can be got, including the significance of different variables. Then the significance can be summarized:



|  |  |
| --- | --- |
| Week | 0.0001417 |
| Day\_of\_Week | 0.0012343 |
| Backup\_Start\_Time | 0.0009629 |
| Work\_Flow\_ID | 0.0023858 |
| File\_Name | 0.0010034 |
| Backup\_Time | 0.0702907 |

Figure.2 Significance of the different variables

From the table, it can be found that the Backup Time matters most, and the Week matters least.

The Fitted values and actual values scattered plot over time is shown in Fig.3 below. Here we choose the best fitting coefficients among the 10 folds.

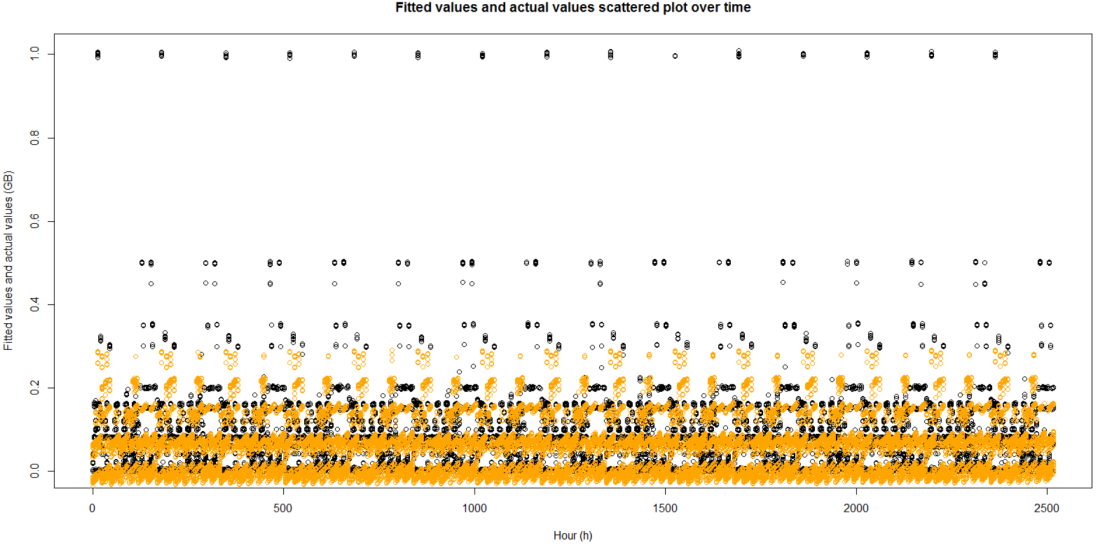


Figure.3 The Fitted values and actual values scattered plot over time

And the residuals versus fitted values is plotted in Fig.4, where difference is: Actual Value - Predicted Value:

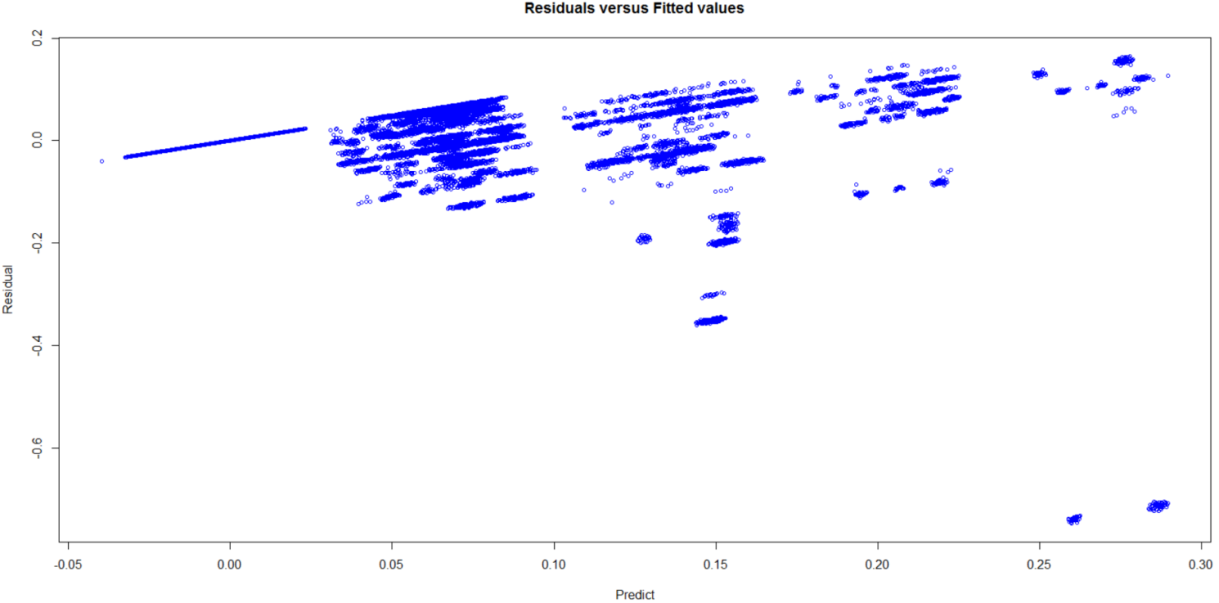


Figure.4 Residuals versus fitted values

*2(b) Random Forest Regression*

First of all, we chose a set of Trees and Depths (due to time limited, we didn't choose many groups, just 8\*8 groups). Then for each pair of (nTree, Depth), we implement 10-fold cross validation and calculate the average RMSE. So finally we got a RMSE list containing 64 members. The data is listed below. It can be seen that when Depth=4 and nTree=50, we got the best RMSE, which is **0.01032132**.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tree\Depth | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 |
| 20 | 0.01078695 | 0.01104907 | 0.01086188 | 0.01083738 | 0.01085715 | 0.01060243 | 0.01080598 | 0.01061919 |
| 25 | 0.01043942 | 0.01074283 | 0.01082893 | 0.01061539 | 0.01061464 | 0.01082291 | 0.01052495 | 0.01085184 |
| 30 | 0.01054020 | 0.01069876 | 0.01052240 | 0.01052712 | 0.01059375 | 0.01060496 | 0.01060165 | 0.01066098 |
| 35 | 0.01054670 | 0.01079748 | 0.01052471 | 0.01066839 | 0.01055420 | 0.01047314 | 0.01051030 | 0.01065190 |
| 40 | 0.01054080 | 0.01041203 | 0.01059019 | 0.01045204 | 0.01037800 | 0.01041274 | 0.01044771 | 0.01049356 |
| 45 | 0.01033435 | 0.01054279 | 0.01040735 | 0.01051046 | 0.01046619 | 0.01039590 | 0.01044319 | 0.01060533 |
| 50 | **0.01032132** | 0.01064598 | 0.01056808 | 0.01052528 | 0.01045027 | 0.01039230 | 0.01033935 | 0.01041479 |
| 55 | 0.01048599 | 0.01039060 | 0.01038753 | 0.01052176 | 0.01049313 | 0.01050612 | 0.01039492 | 0.01058316 |

After comparison with linear regression, it's obvious that using *randomForest* will help get **better** RMSE.

Then we analyzed the parameter significance using summary(fitting). It shows that the "Week" weighs very little compared to other parameters. So it can be concluded that under this model, the RMSE trend will repeat every week, that is roughly *160* hours.

*2(c) Neural Network Regression*

We used the **pybrain** library (one of the most commonly used for neural network in Python) to implement the neural network regression model. As a result of the 10-fold cross validation, the best Root Mean Squared Error (RMSE) we can get is 0.118428240362.

The major parameters of our neural network model are the number of hidden nodes, the number of hidden layers and the learning rate. Picking a correct number for hidden layers and a correct number for the number of nodes in the hidden layer is very important. The introduction of hidden layer(s) makes it possible for the network to perform non-linear regression, but higher dimensionalities of the hidden layers are prone to over-fitting. In order to secure the ability of the network to generalize and predict, the number of hidden nodes has to be kept as low as possible, otherwise our model will not perform well on data that are not part of the training set. Having a optimal learning rate is also important, because too low a learning rate may cause the neural network to be stuck in local minima, and too high a learning rate may cause it to not converge at all.

With the help of the 10-fold cross validation, we tuned these parameters and chose the number of hidden nodes and layers that gives the lowest RMSE. Having 2 hidden layers (0.118428240362) gave us lower RMSE than having 1 hidden layer (0.141693085531) and 3 hidden layers (0.122700481986); Having 10 hidden nodes gave us the best RMSE (0.118428240362), compared to having 7 hidden nodes (0.132536116187) and having 15 hidden nodes (0.14289441).

*Question.3*

*3(a) Backup Size for each Workflow*

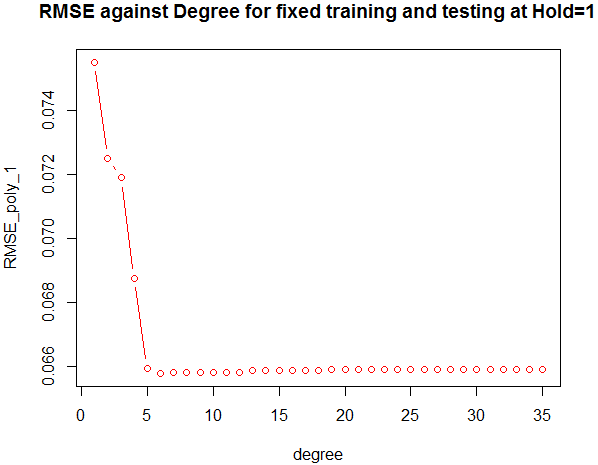
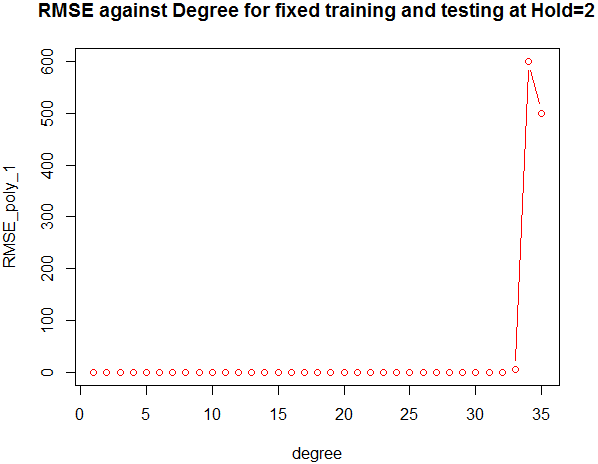
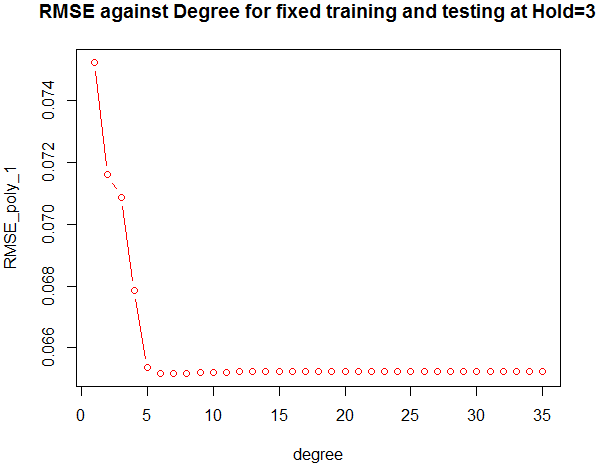
In this question, for each Workflow, we implemented the linear regression with 10-fold cross validation, just like what we did in 2(a). Then the Backup Size for each of the Workflow is summarized below:

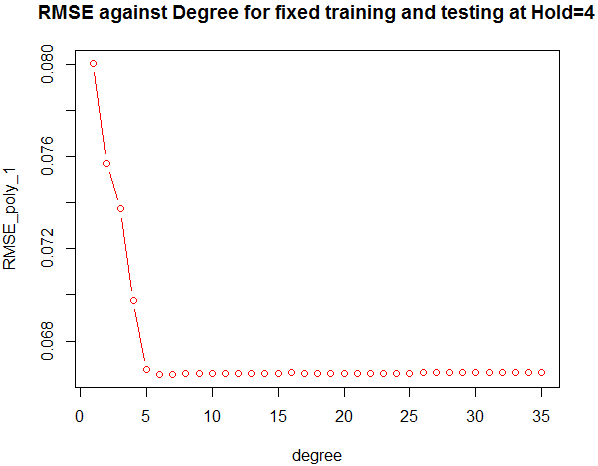
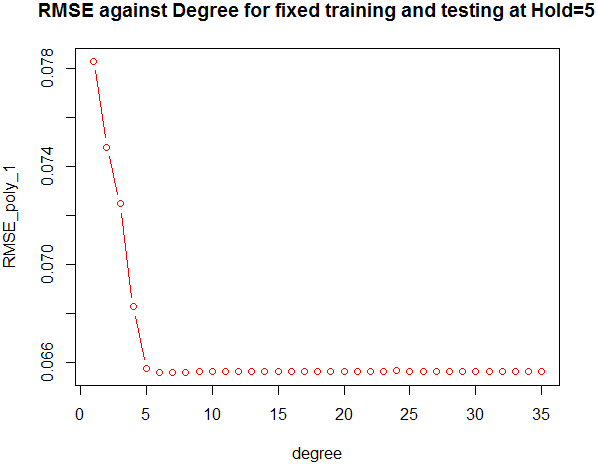
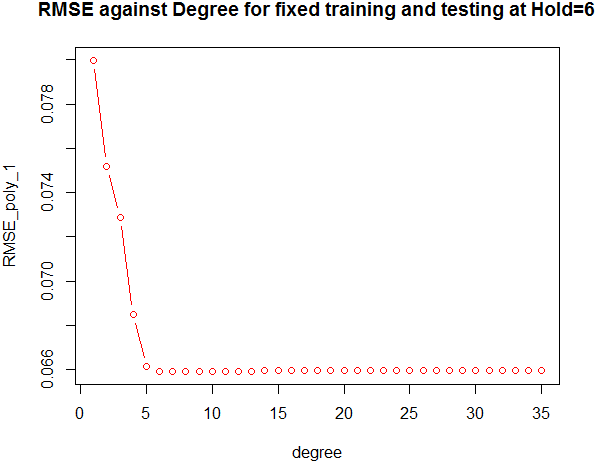
|  |  |
| --- | --- |
| Work Flow | RMSE |
| Work Flow 1 | 0.02946 |
| Work Flow 2 | 0.10376 |
| Work Flow 3 | 0.02550 |
| Work Flow 4 | 0.00588 |
| Work Flow 5 | 0.08417 |

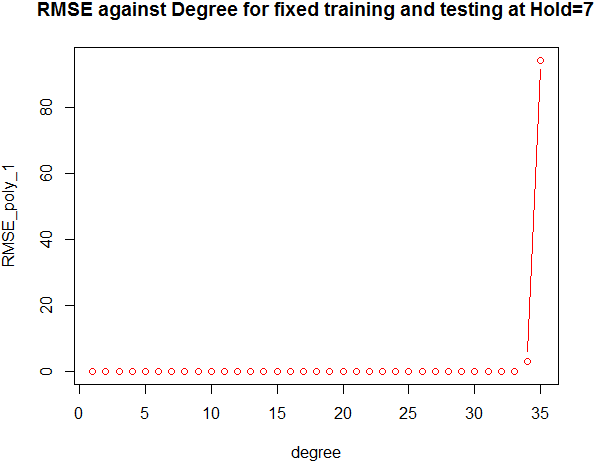
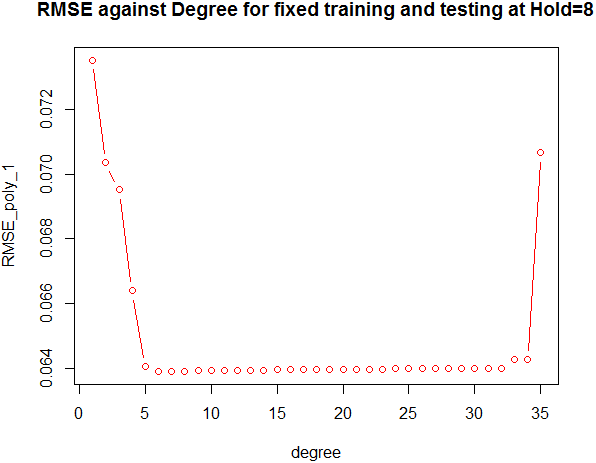
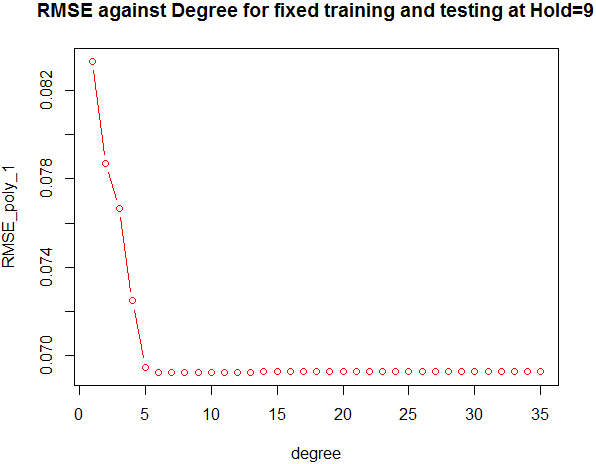
Compared to the **0.0789572315048197** which is calculated in 2(a), the RMSE for some of the Workflows is increased, but not for others. So the fit is not improved a lot.

*3(b) Backup Size for each Workflow*

First, the RMSE versus the Degree of Polynomial is plotted for each of the training mode (fold = 1,2,3...10). The results are shown in Fig.5 below:

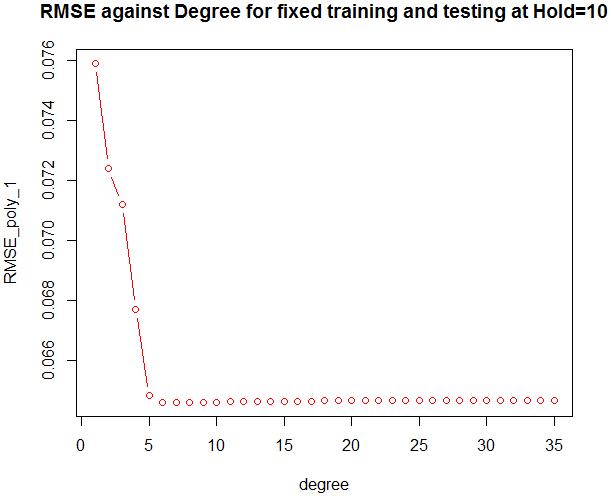


Figure.5 The RMSE versus the Degree of Polynomial

Then, the average RMSE versus Degree of Polynomial is plotted as shown in Fig.6 below. From the figure, it can be roughly concluded that the threshold is **34**.

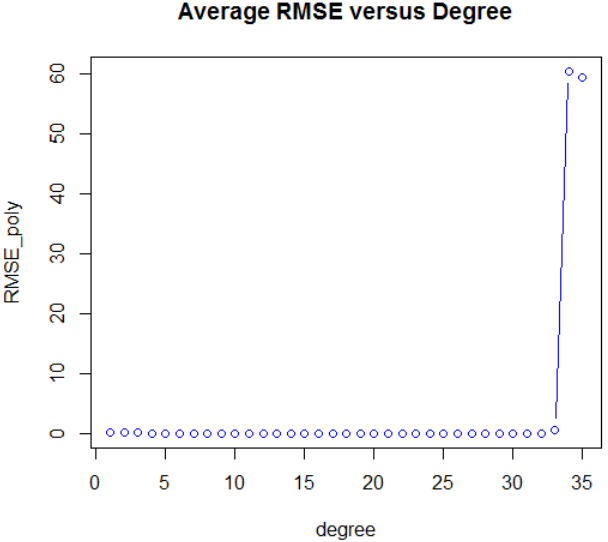


Figure.6 Average RMSE versus Degree of Polynomial

**Boston Housing Dataset**

*Question.4*

Here, we used *train* function from *caret* package to fit a linear regression model with MEDV as the target variable and the other attributes as the features and ordinary least square as the penalty function.

After runing the R script we wrote, using command “summary(lmFit)” in the R console, we can see the value of estimated coefficient for each predicator we obtained from the ordinary linear regression model, as shown in the Fig.7. According to the value of estimated coefficient for each predicator, we can analyze the significance of different variables. For example, NOX varaible could have the most significant influence on MEDV. RM, CHAS and DIS variables could also have significant influence on MEDV. Compared with NOX, RM, CHAS and DIS variables, the influences of other viarables on MEDV are not that significant.

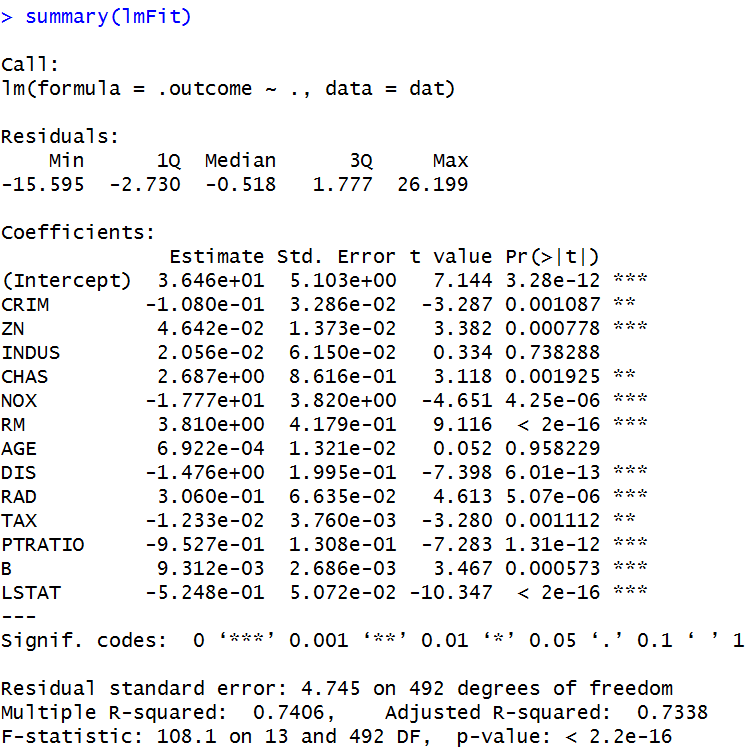


Figure.7 Screenshot of linear regresion model summary

Using command “lmFit” in the R console, we can get the averaged RMSE, as shown in the following Fig.8. Here, the average RMSE is 4.771103.

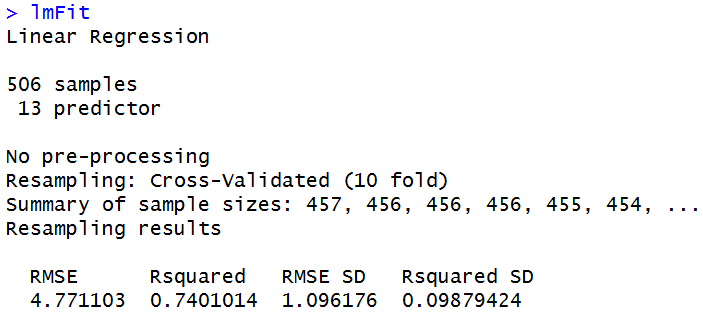


Figure.8 Screenshot of linear regresion model default summary

Fig.9 shows the “fitted values and actual values scattered plot over time”. From Fig.9, we can see that most predictions (i.e. fitted value) are closed to the actual values (except for some points whose actual value of MEDV is 50 in the dataset), which indicates that our ordinary linear regression model could fit the data.

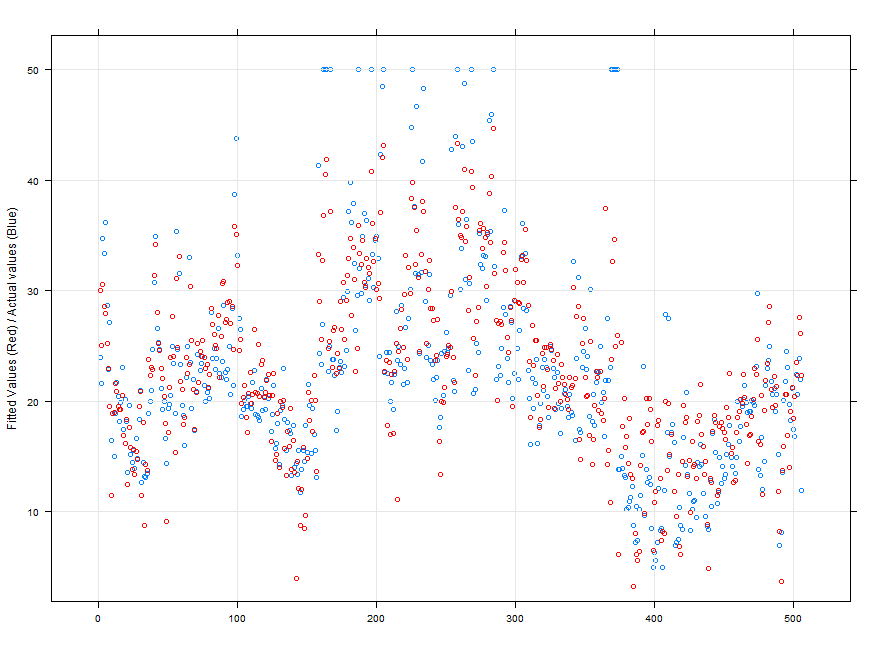


Figure.9 Fitted values and actual values scattered plot over time

Here, we also provided another plot, i.e. actual values versus fitted values (as shown in Fig.10) , which we can use to easily assess how close the predictions are to the actual values. From Fig.10, we can see that most of points are closed to “straight line y=x” (except for some points whose actual value of MEDV is 50 in the dataset), which also intuitively indicates that our ordinary linear regression model could fit the data.

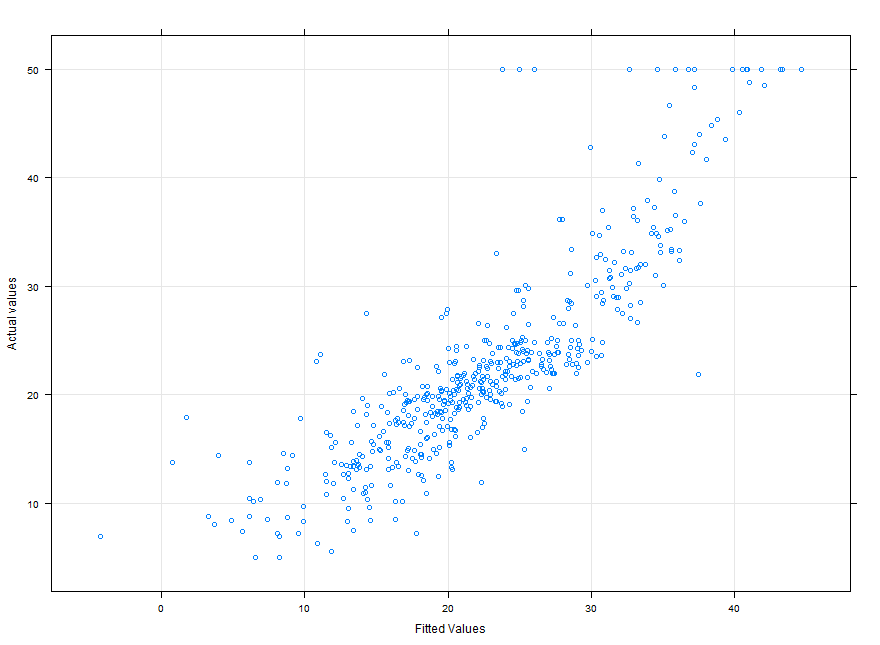


Figure.10 Actual values versus fitted values for the boston housing dataset

It is also important to check model assumptions such as the residual distribution. Here, we plotted the residuals versus the fitted values, as shown in Fig.11. The plot shows a random cloud of points (except for some points whose actual value of MEDV is 50 in the dataset), which means there are no major terms missing from the model or significant outliers.

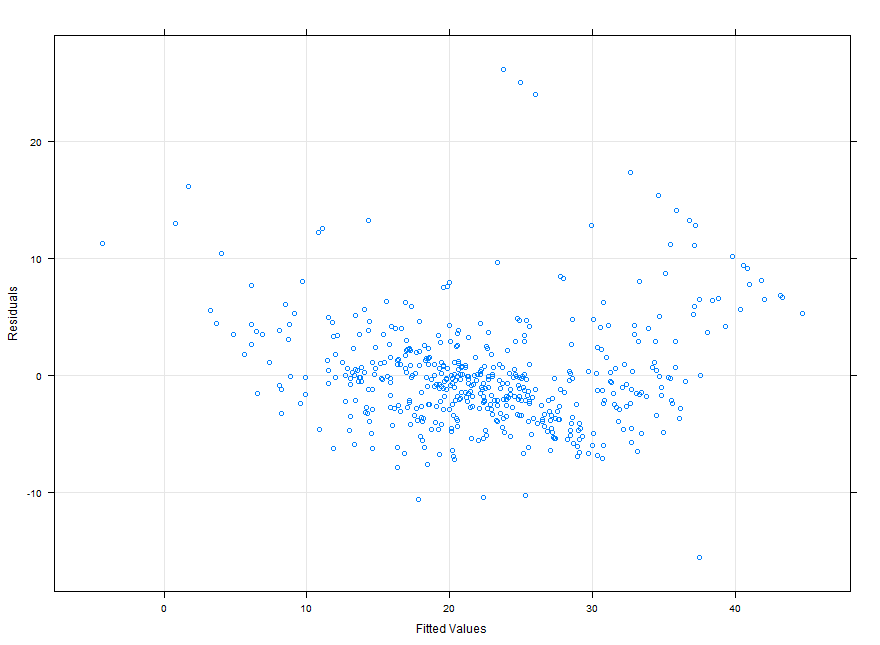


Figure.11 Residuals versus fitted values for the boston housing dataset

Then we tried to fit a more complex regression model to the dataset by using polynomial function of each predictor. The polynomial function we used is f(n)=xn+xn-1+…+x2+x+b, where x is the predictor (i.e. the variable that we use as features), n is degree of polynomial function and b is intercept. During each iteration, we use same degree for all predictors. Fig.12 shows our result of polynomial regression model exploration.

|  |  |
| --- | --- |
| **The degree of**  **polynomial function** | **Average RMSE** |
| **1** | 4.771103 |
| **2** | 3.978978 |
| **3** | 4.011302 |
| **4** | 3.979028 |
| **5** | 5.901913 |
| **6** | 10.74559 |
| **7** | 21.04673 |
| **8** | 510.6772 |
| **9** | 2483.884 |
| **10** | 904.8164 |

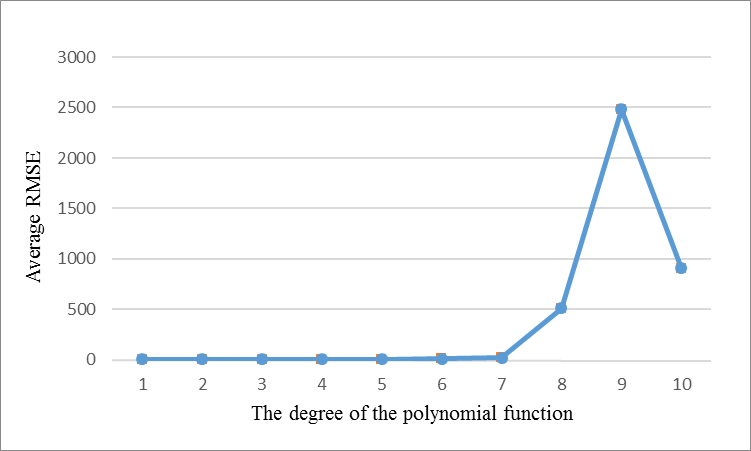


Figure.12 Average RMSE of the trained models against the degree of the polynomial function

From Fig.12, we can see that when the degree equals to 2, 3 and 4, the average RMSE is around 4, which is smaller than the RMSE that we got from linear regression model. In addition, when the degree is greater than 7, the RMSE increase significantly and thus the error of the model gets worse. So, according to our exploration, the optimal degree is 2.

*Question.5*

*5(a) Tune the complexity parameter*

Here, we still used *train* function to tune the ridge regression model by specifying *method = ”ridge”*. The parameter *lambda*,i.e. the parameter α in the problem spec, controls the ridge regression penalty. We set the *lambda* in the range {0.1, 0.01, 0.001} in our R script, as specified in the problem spec.

After runing the R script we wrote, using command “ridgeFit” in the R console, we can see the summary information of the ridge regression model we built, as shown in Fig.13. The best RMSE is 4.799581. We can also see that the RMSE slightly increases when the parameter *lambda* (ridge regression penalty) increases.

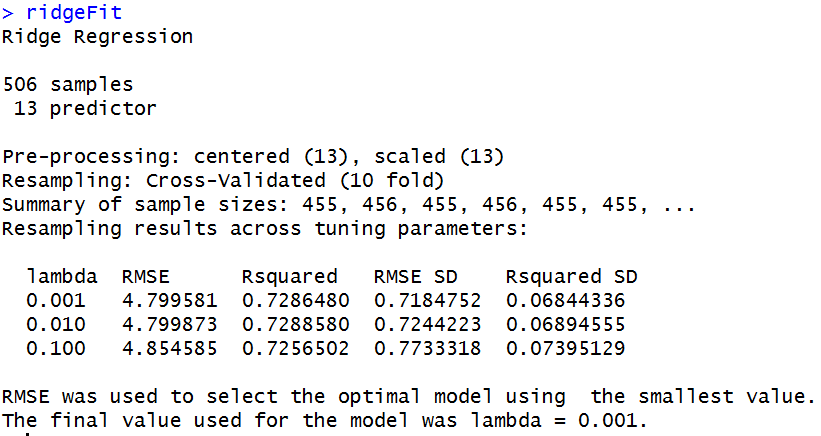


Figure.13 Screenshot of ridge regresion model summary

*(b) Repeat the previous part for Lasso regularization*

Here, we still used *train* function to tune the lasso regression model by specifying *method = ”enet”*. Actually, *enet* model combines the two types of penalties: ridge regression penalty and lasso regression penalty. In order to fit the lasso regression model, we set *lambda* to 0, since the *lambda* controls the ridge regression penalty. The parameter *fraction*, i.e. the parameter α in the problem spec, controls the lasso regression penalty. We set the *fraction* in the range {0.1, 0.01, 0.001}, as specified in the problem spec. In addition, we also tried *fraction*=1 to check if the Lasso regression model is correct.

After runing the R script we wrote, using command “lassoFit” in the R console, we can see the summary information of the lasso regression model we built, as shown in Fig.14. The best RMSE is 7.664197 if the *fraction* is in the range {0.1, 0.01, 0.001}. We can also see that the RMSE decreases when the parameter *fraction* increases.

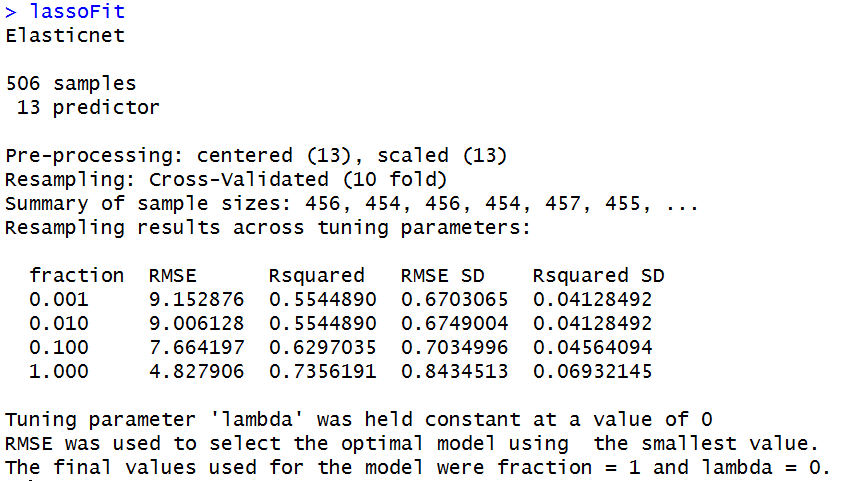


Figure.14 Screenshot of lasso regresion model summary